Assignment 2

CS-GY 6033 INET Fall 2024

Due date: 11:55pm on **October 12th 2024 on Gradescope**.

Instructions:
Below you will find the questions which make up your homework. They are to be written out (or typed!) and handed in online via Gradescope before the deadline.

Question 1: Hashing

.

(a) 5 points Consider a hash table implemented as an array, $T[]$, indexed from $0 \ldots 16$. Hashing is carried out using double hashing, with $h_1(k) = k \mod 17$ and $h_2(k) = k^2 + 1 \mod 17$. Show the insertion of: $45, 99, 32, 96, 25, 36, 83, 27, 21, 49, 51, 8$, or explain why it is that certain keys are not inserted.

(b) 5 points Repeat the above, but where hashing is carried out using the quadratic hash function $h(k, i) = k + 2i + 3i^2 \mod 17$. Show the insertion of the keys from part (a), or explain why it is that certain keys are not inserted.

(c) 3 points Repeat the hashing once again, this time using function $h(k) = k^2 \mod 17$, where collisions are resolved with chaining.

(d) 3 points Of the above three hash tables, which method requires the most probes in order to **insert** key 15 in the table?

(e) 4 points Consider the same hash table from parts (a), which has size 17 and collisions are resolved with double hashing. Suppose a new key k is inserted into the table using the double hashing technique from part (a). How many insert attempts should be made, before the conclusion is made that no free spot will be found? Is it possible that a free spot exists, even though the double hashing method does not find it?

Question 2: Selection

.

(a) 6 points Carry out the *Select* algorithm on set below using $k = 14$ (return the element of rank 14). Show your steps! You do not need to show the steps of the partition algorithm. Instead, ensure that the output of the partition algorithm maintains the elements in their original relative order. You must carry out *all* recursive calls to the Select algorithm. You may assume that the base case of the Select algorithm is any input of at most 5 elements, in which case it returns the element of rank k.
**When taking the median of an even number of elements, use the lower median by default*

$$25, 37, 52, 14, 89, 35, 83, 53, 31, 86, 99, 46, 66, 34, 22, 2, 8, 90, 30, 68,$$

$$21, 17, 84, 29, 77, 45, 33, 41, 19, 53, 42, 93, 23, 18, 91$$

(b) 6 points Repeat the above, using the Randomized-select algorithm. Seeing as you don't have a random-number generator on hand, you will select your "random" pivot with bias: always select the *first* element in your array as your random pivot. As above, you don't need to show the execution of the partition algorithm, but ensure for consistency that the output of the partition algorithm maintains the elements in their original relative order.

(c) 8 points Suppose we alter the Select(k) algorithm from class, so that it groups the elements into groups of size 6 instead of groups of size 5. The main steps of the algorithm are:

1. Sort each group of size 6. If there is only one group, return the element of rank k.
2. Identify the 3rd smallest element of each group (the lower median). Make a recursive call to find the median of the medians. Call this element x
3. Partition the input into those that are smaller and those that are larger than x. Let r be the rank of element x.
4. If $k = r$,return x
5. If $k < r$, recurse on the set of smaller elements. Otherwise recuse on the larger elements.

Your job: Determine the runtime of each of the above steps. Write the recurrence relation for this algorithm. Show that the overall runtime of this version of Select is $O(n)$ using the substitution method.

Question 3: Heaps

.

(a) 10 points Below are three different methods for building a max-heap. Suppose that $A[1, 2, \ldots, n]$ initially contains n elements, which are not necessarily in heap format. You may assume that the value of $A.heapsize = n$. For each method, justify whether the method correctly. builds a max-heap. If so, justify your answer. If not, provide an example showing it doesn't work.

METHOD 1:
Initial call: Make-Heap1(A, n)
Make-Heap1(A, i)
    If $i > 1$
        Bubble-up(A,i)
        Make-Heap1(A,$i - 1$)

METHOD 2:
Initial call: Make-Heap2(A, n)
Make-Heap2(A, i)
    If $i \geq 1$
        Bubble-down(A,i)
        Make-Heap2(A,$i - 1$)

METHOD 3:
Initial call: Make-Heap3($A, 1$)
Make-Heap3(A, i)
    If $i \leq A.heapsize$
        Bubble-up(A,i)
        Make-Heap3(A,$i + 1$)

(b) 8 points A *min-max-heap* is a complete binary tree containing alternating *min* and *max* levels. The root node is the **minimum** of all nodes in the tree. The nodes at the next level are the largest nodes of their subtrees. For each level after that, a node that is on a min level is the minimum of all nodes in its subtree, and a node that is on a max level, is the maximum of all nodes in its subtree. Below is an example of a min-max tree. Assume that the elements are stored in array $A[]$ as for usual heaps, where $A.heapsize$ indicates the size of the heap. Your job is write the pseudo-code for a procedure that **inserts** into this type of heap. Call your procedure MinMaxInsert(A, k), which inserts k into the heap, and carries out the necessary swaps in order to maintain the min-max heap property.

(c) 6 points Suppose A is a sorted list of exactly n elements, sorted in decreasing order. Suppose B a max-heap of exactly n elements. Two students are given the task of creating a sorted list of the combined elements from set A and B. Below are their approaches:

Student 1: Add all elements of A and B to a new array C of size $2n$. Then uses bottom-up heap building, resulting in a max heap in array C. Next, run HeapSort on array C.

Student 2: Repeat until A and B are empty: Compare first element of A with element $B[1]$. If top element of A is larger, extract it to the output list. If $B[1]$ is larger, call Delete-max, and store to the output list. If at some point B is empty, simply concatenate all remaining elements of A to the output list. If at some point A is empty, continue calling Delete-max on B until it is empty.

Determine the worst-case runtime for each method above. Is there one student with a more efficient approach?

Question 4: Lower Bounds and Linear time Sorting

.

(a) 10 points Let $A[]$ be an array of n real (decimal) numbers, uniformly distributed from 1 to 100. A student would like to print out the smallest t $\sqrt{n}$ numbers in **increasing** order. Consider the three approaches below:

OPTION 1

1. $k = \sqrt{n}$.
2. $p =$Select$(A, 1, n, k)$.
3. Loop through $A[]$ and store all items that are less than or equal to p, in a list L
4. Sort list L in increasing order, using QuickSort.
5. Print L

OPTION 2

1. Set-up Bucket sort using 10 equally-sized buckets in the range 1 to 100. Bucket 1 covers the range 1 to 10 inclusively.
2. Distribute the points into buckets.
3. Run insertion sort on the elements of bucket 1
4. Print out the sorted elements from bucket 1.

OPTION 3

1. Loop through input $A[]$ and set m to be the maximum element found
2. Use Counting sort with maximum-sized element m
3. Print out the first $\sqrt{n}$ elements from the resulting sorted array

Your Job:

For each option, determine if the procedure correctly solves the problem.

Determine the **Expected runtime** of each approach, and justify your answer.

Which option is expected to be asymptotically faster? Or are they expected to be asymptotically equivalent?

(b) 4 points Draw the decision tree for the in-place partition algorithm on four elements.

(c) 4 points A car license consists of 6 characters, where the characters alternate between alphabetic characters (A to Z) and numerical characters (1 to 9). An example of a valid license is $B1C3X4$. Explain how to sort a set of n licenses using Radix sort, and justify the runtime.